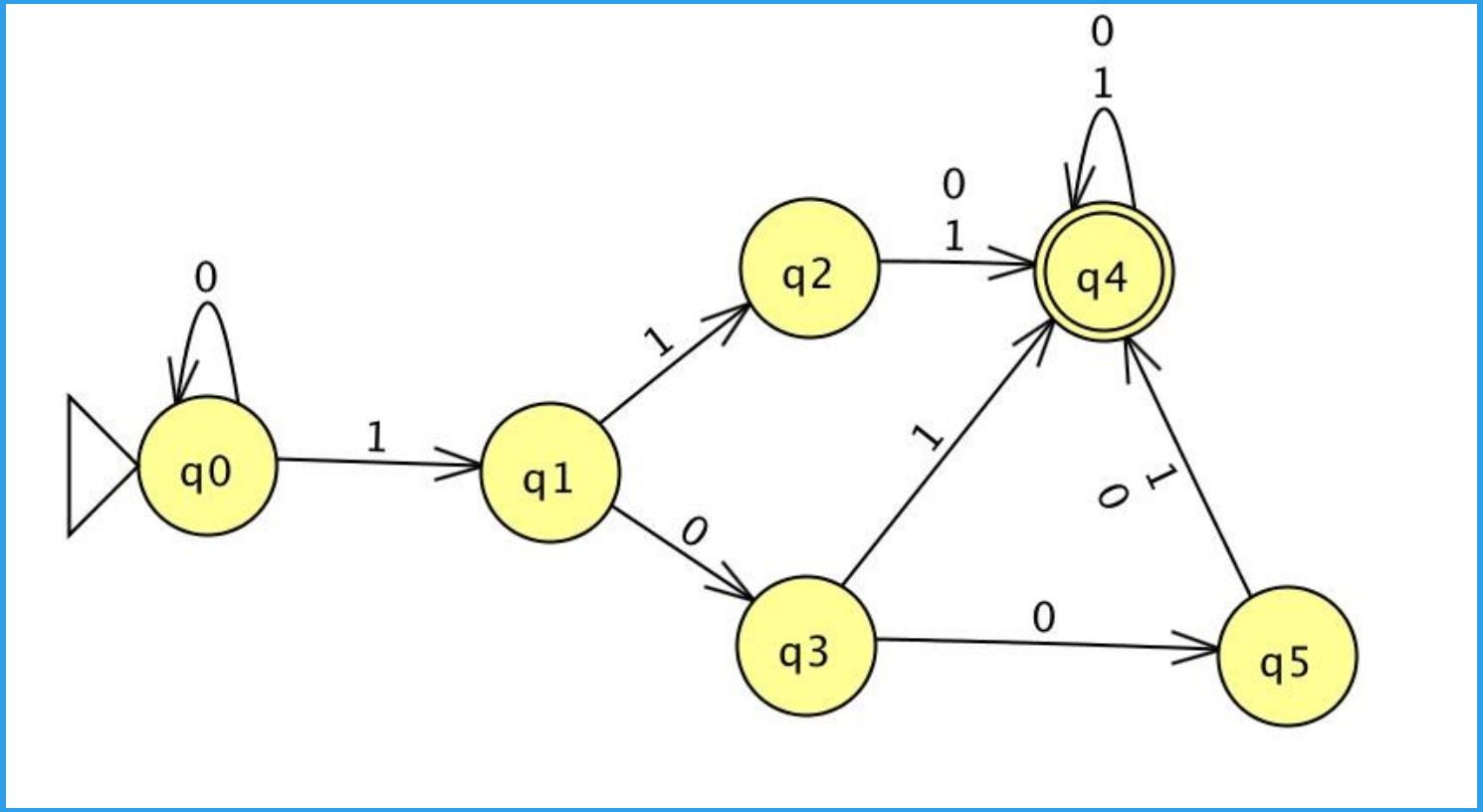
Theory of Computation and Programming Languages



NFAs

CS51 – Fall 2025

Formal definition of DFAs

- A Deterministic Finite Automaton (DFA) is a finite state machine that accepts or rejects finite strings of symbols and produces the same unique computation for each unique input string. For any given finite input string, the DFA will halt and either accept or reject the string. A DFA, M, is said to recognize a language, $L(M)$, which is the set of all strings that M accepts.
- Formally, a DFA is described by a 5-tuple $(Q, \Sigma, \delta, q0, F)$
  - Q is a finite set of states
  - Σ is a finite set of input symbols also known as the alphabet
  - δ is a state transition function ($\delta : Q \times \Sigma \rightarrow Q$)
  - $q0$ is the start state ($q \in Q$)
  - F is a set of accept states ($F \subseteq Q$).

Computing

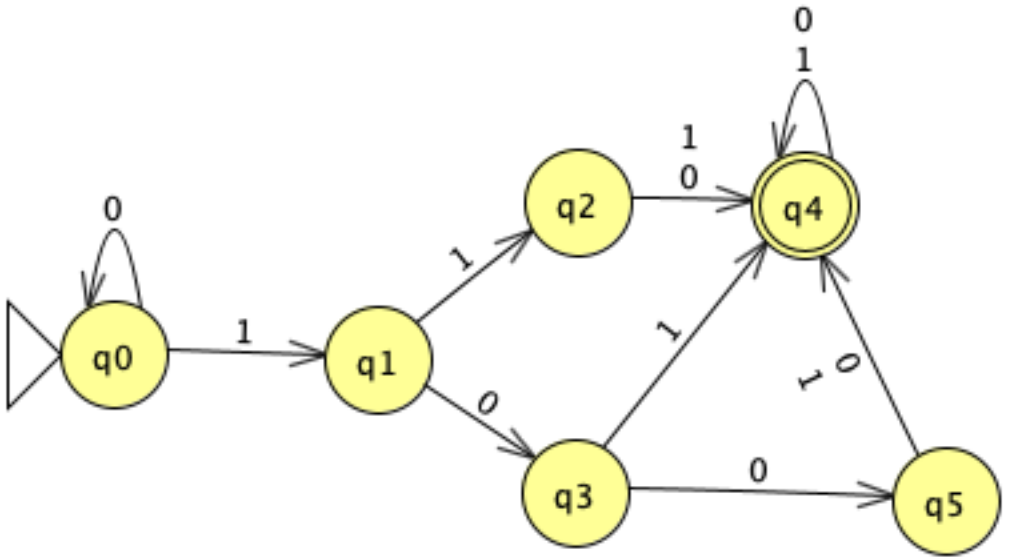
- We have a string as input on a tape.
- We start at the beginning of the string.
- We read a symbol from the tape and transition to the state indicated by the model.
- If we end in a final state (i.e. when we get to the end of the string we are in a final state), we accept the string.
- Otherwise, if when we get to the end of the string on the tape and we're in a non-final state we reject.
- All strings accepted by a DFA define a language.

DFAs over numbers

- We can use any alphabet we want
- If we use 1's and 0's we can interpret them as binary numbers!

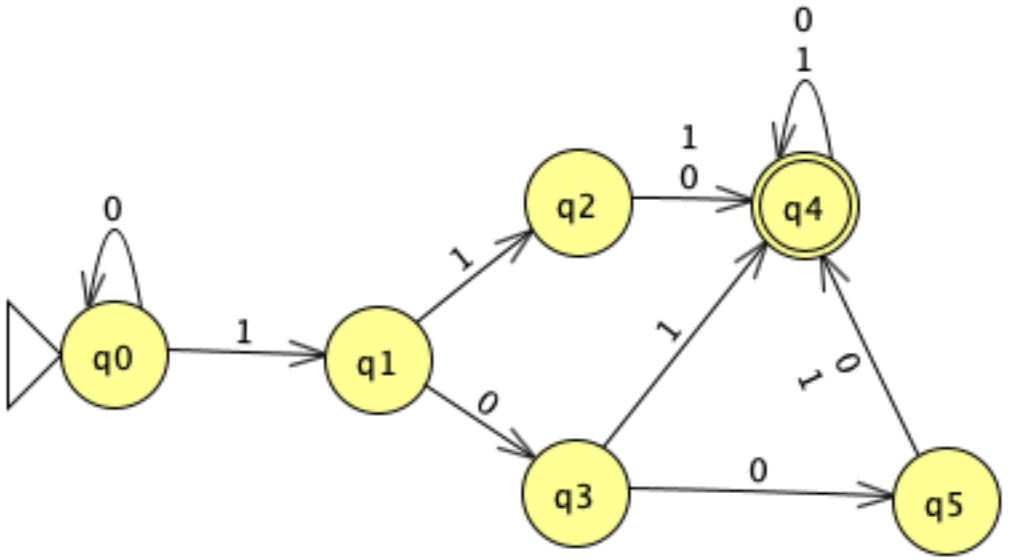
Practice time

- Assuming we work on 0s and 1s that are interpreted as bits, what does the following DFA do?



Answer

- Determines if an input string of 0s and 1s, when interpreted as a binary number, is greater than or equal to 5.

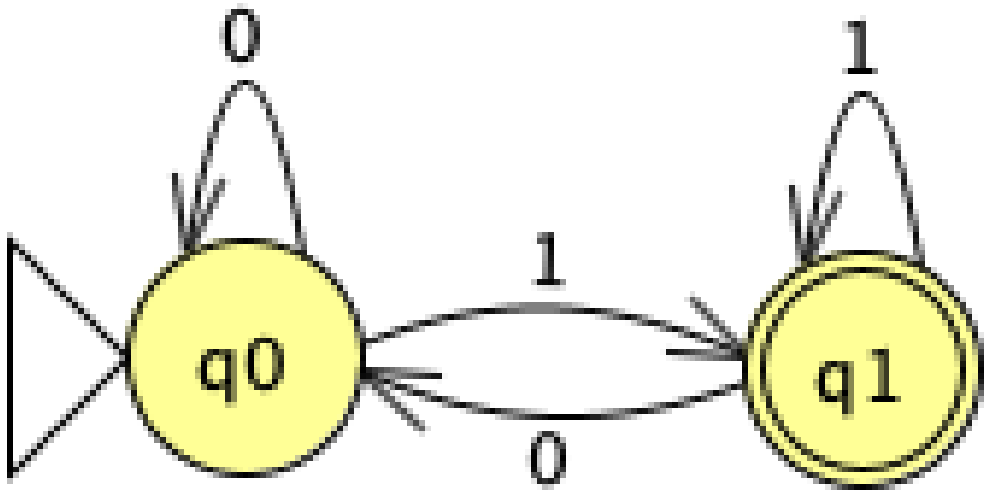


Practice time

- Can you write a DFA that determines if a number is odd?

Answer

- Can you write a DFA that determines if a number is odd?
- Remember, odd numbers end in 1.



Deterministic finite automata

- A Deterministic Finite Automaton (DFA) is a finite state machine that accepts or rejects finite strings of symbols and produces the **same unique computation** for each unique input string. For any given finite input string, the DFA will halt and either accept or reject the string. A DFA, M, is said to recognize a language, $L(M)$, which is the set of all strings that M accepts.
- This unique computation where we know what state we can go to given an input is what the term deterministic means.
- In particular, for any state of a DFA, the state transition function specifies exactly one state for each input symbol in the alphabet.

Non-deterministic finite automata

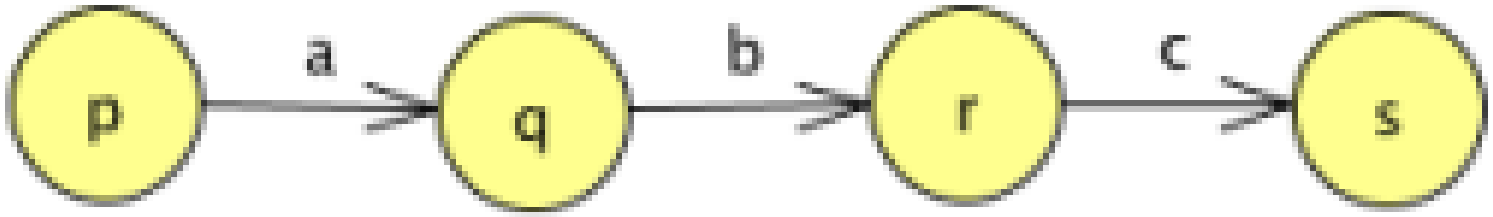
- Non-deterministic finite automata (NFAs) are generalizations of the DFA concept.
- In an NFA, for a given state and input symbol, we can go to **zero**, one or **more** states (rather than just a single one for DFAs).
- Tend to be a bit easier to create than DFAs
- An NFA accepts a string if *some* path exists through the DFA based on the input string that end in the final state.

NFA Example 1

- Suppose our alphabet is {a, b, c} and we are interested in the set of all words that contain the string *abc* or that end with the string *ac*.
- Some examples of such words are:
  - w1 = aaabc
  - w2 = babcabcac
    - Note that w2 has the substring abc in multiple locations *and* also ends with ac.
  - w3 = aaac
- On the other hand, the string w4 = abbccaab is not in the desired set.
- Let's try to construct a finite machine that accepts such words.

NFA Example 1

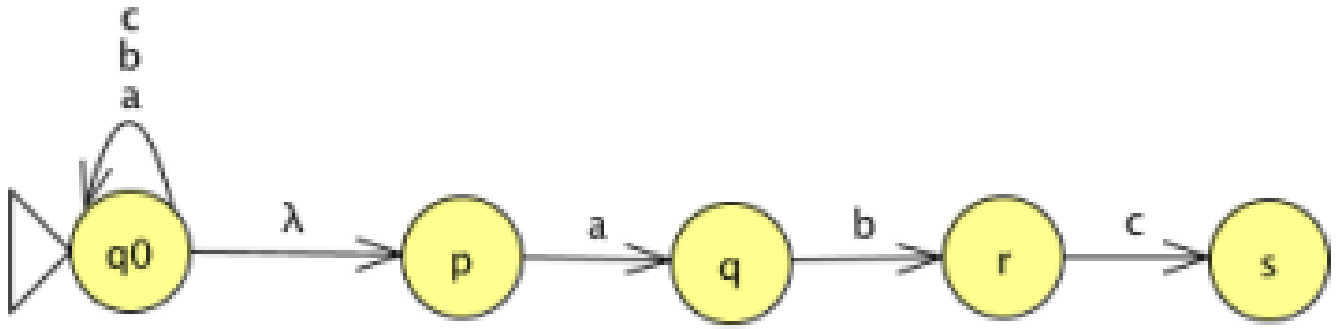
- Suppose we want the string *abc* anywhere in the word.
- We want the following configuration to be part of our finite machine:



- We can move from p to s with the string abc.
- Since this occurrence of abc can be anywhere in the word, we want to be able to *reach* the state p at any step.
- Also, once we reach s, we want to accept the word with any combination of input symbols following this occurrence of abc.

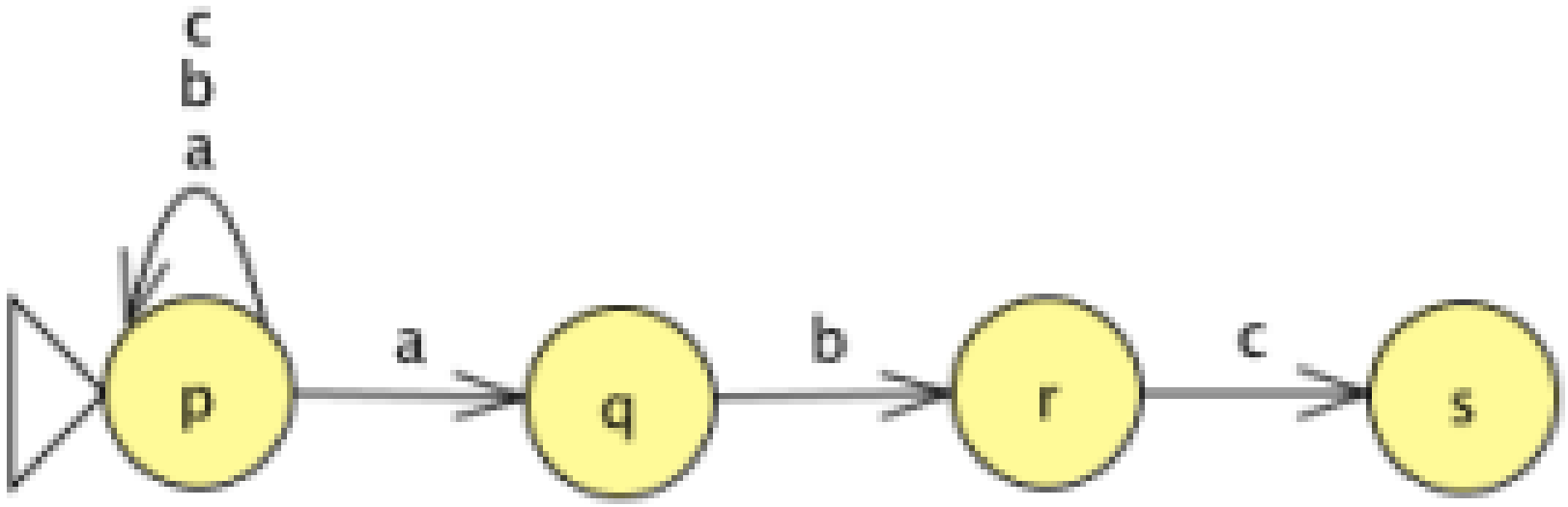
NFA Example 1

- We introduce an initial state q0.
- The transitions labeled a, b and c at q0 can read any combination of the symbols in the alphabet.
- The transition labeled λ allows one to “jump” from q0 to p without reading an input symbol.
- Thus, we may “guess” where an abc might occur to make this jump.



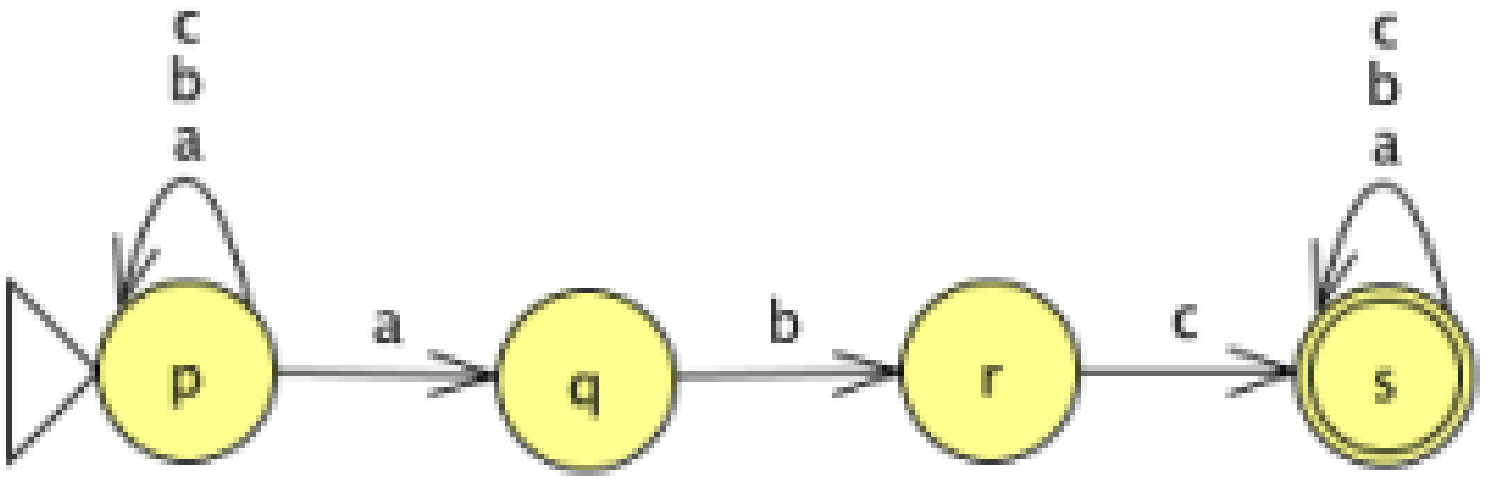
NFA Example 1

- Equivalently without the λ transition:
  - State p is the initial state. We observe that there are two transitions labeled a from the state p. One stays in p while the other goes to q.
  - This nondeterministic behavior allows one to make the jump to the state q when looking for the substring abc.



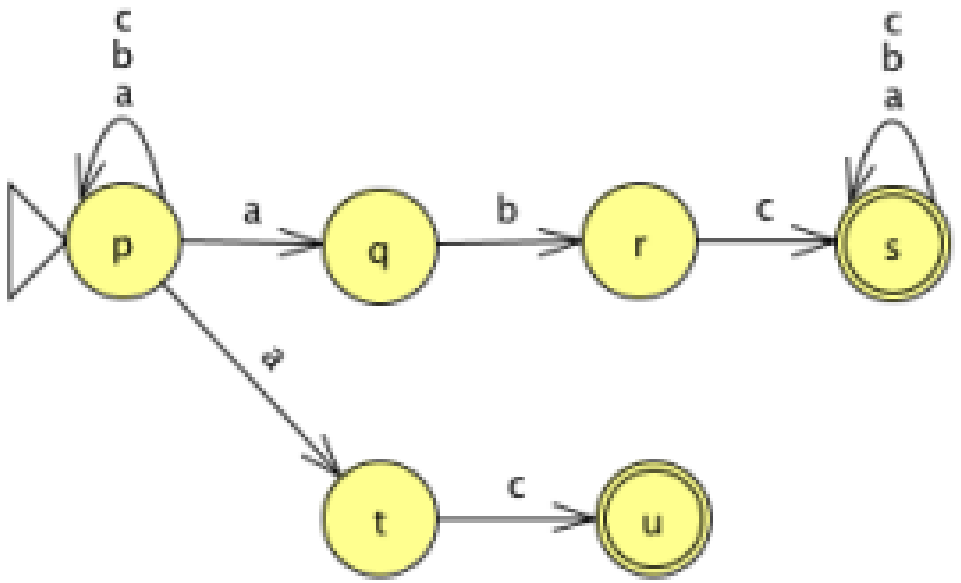
NFA Example 1

- Next, we introduce transitions a, b and c at state s and make s an accept state.
- Thus, any word containing the string abc is accepted.



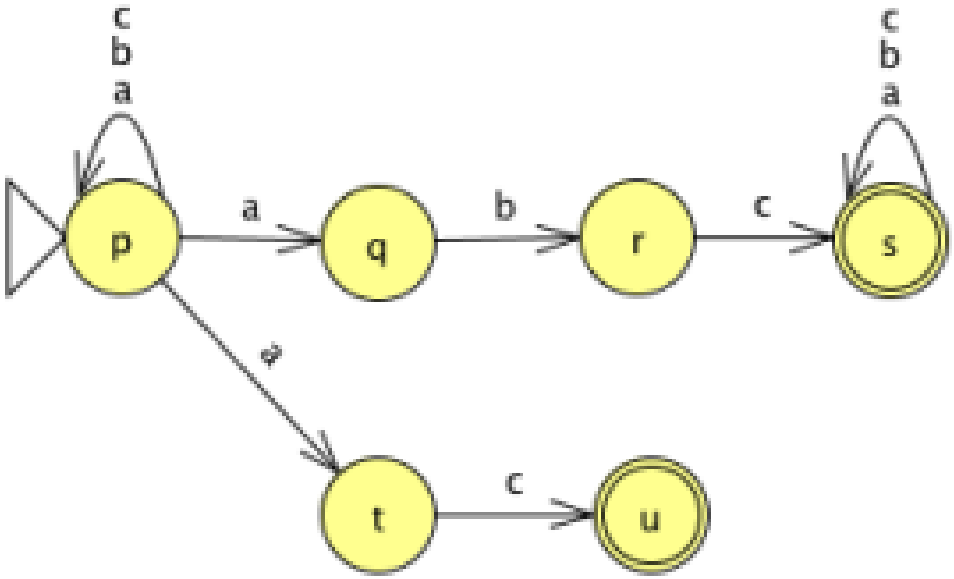
NFA Example 1

- To accept words that end with ac, we introduce two more states and transitions as shown below.
- This is a Nondeterministic Finite Automaton (NFA) that accepts our desired set of words.



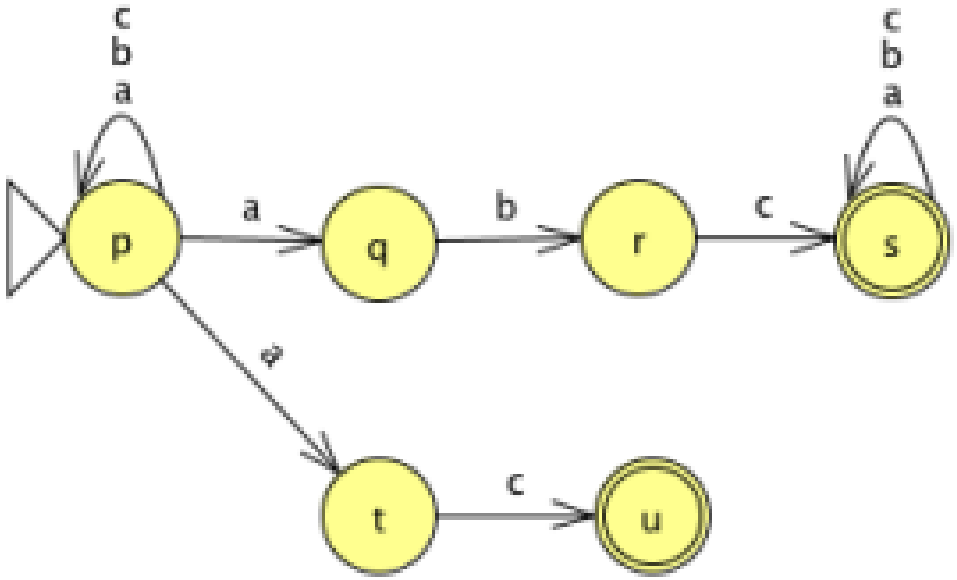
NFA Example 1

- We note several properties of this NFA:
- There are three transitions from state p labeled a. Another way of saying this is that the transition labeled a from state p goes to the set {p, q, t}.
- There are no transitions labeled a or c from the state q. Another way of saying this is that the transitions labeled a and c from state q go to the empty set ϕ.
- There can be transitions labeled λ.



NFA Example 1

- Properties such as above distinguish an NFA from a DFA.
- Another consequence of nondeterminism is that there may be several possible paths for a given input string.
- For example, for the input w = ababc, there are several configurations in the NFA.
- an NFA accepts an input if there is *at least one* accepting configuration.

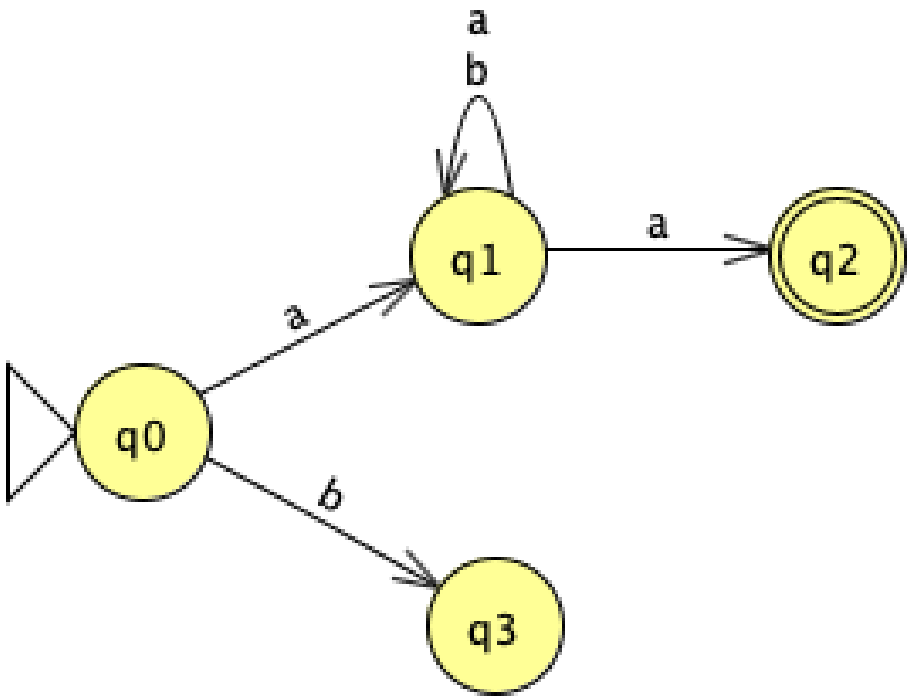


Formal definition of NFAs

- Formally, a DFA is described by a 5-tuple $(Q, \Sigma, \delta, q0, F)$
  - Q is a finite set of states
  - Σ is a finite set of input symbols also known as the alphabet
  - δ is a state transition function ($\delta : Q \times (\Sigma \cup \{\lambda\}) \rightarrow P(Q)$), where $P(Q)$ is the powerset, that is the set of all subsets of Q.
  - $q0$ is the start state ($q0 \in Q$)
  - F is a set of accept states ($F \subseteq Q$).
- An input w is accepted by an NFA if there is at least configuration for w that ends in an accept state, that is a state that belongs to F.

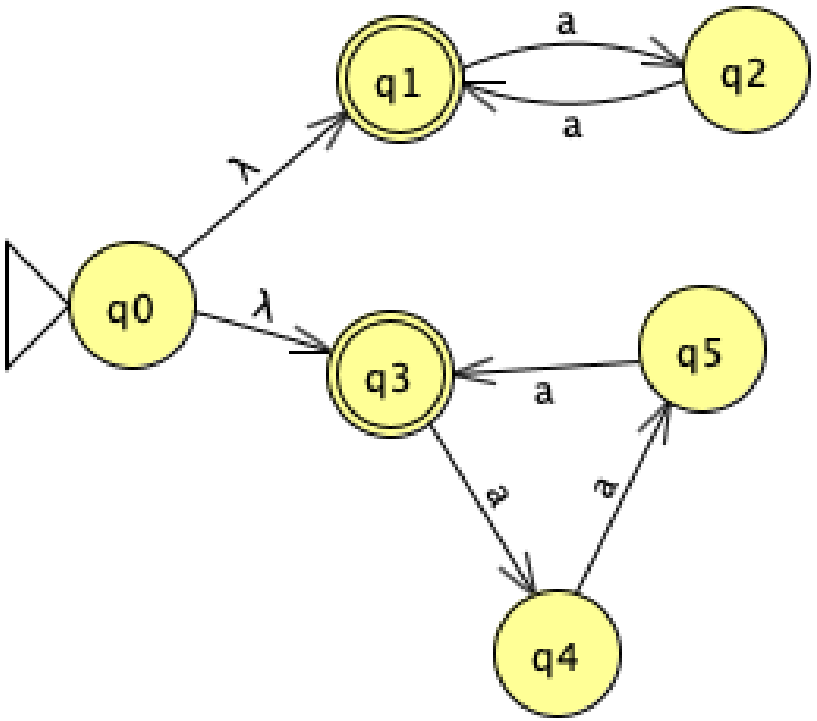
NFA Example 2

- Accepts strings that start and end with a.
- a(a|b)*a



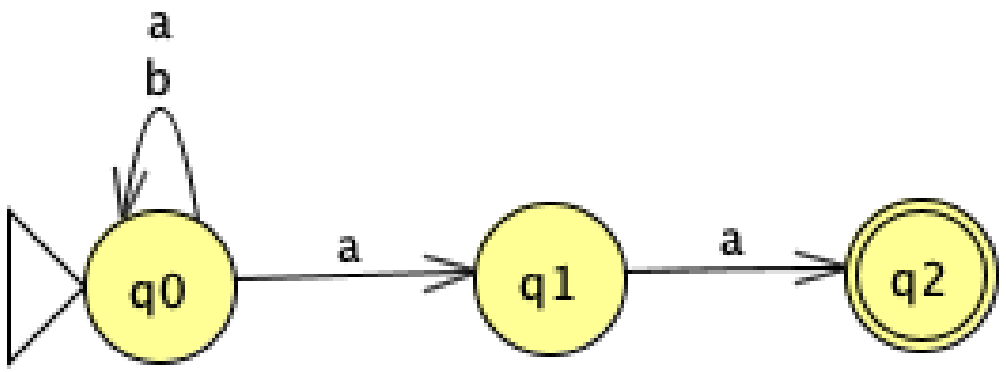
NFA Example 3

- Accepts strings of a's that have lengths divisible by 2 OR 3
- (aa)*|(aaa)*



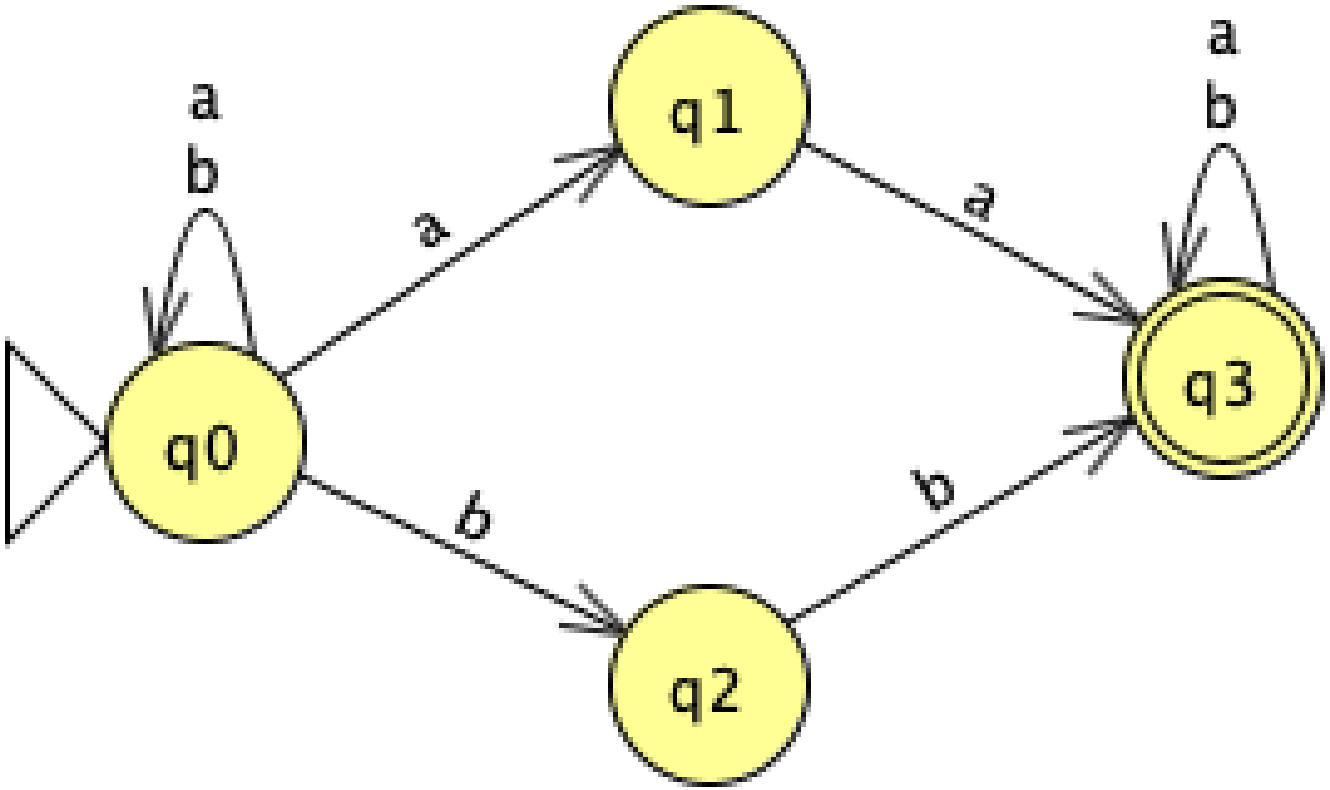
NFA Example 4

- Accepts strings of a's that end in 2 a's
- (a|b)*aa



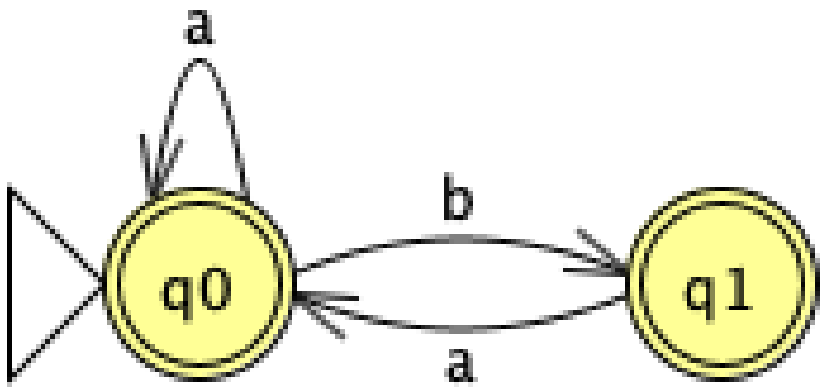
NFA Example 5

- Accepts strings that either have aa or bb as a substring.
- (a|b)*(aa|bb)(a|b)*



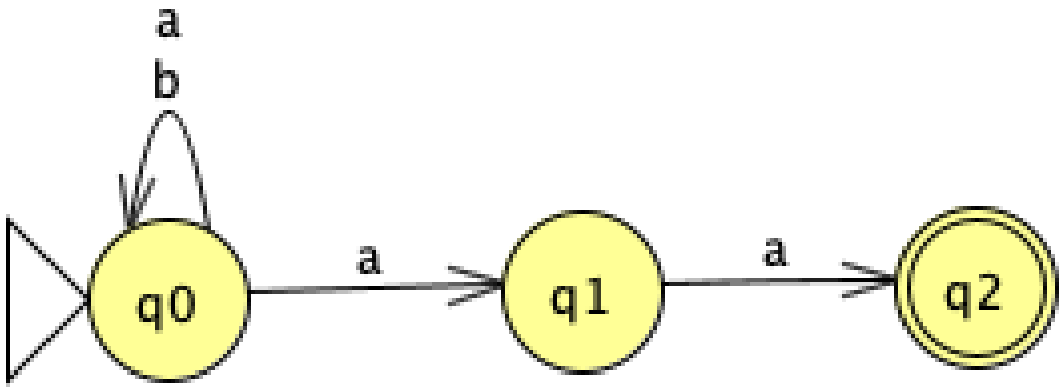
NFA Example 6

- Accepts any string of a's and b's that doesn't have two adjacent b's.
- a*b(aa*b)*a*



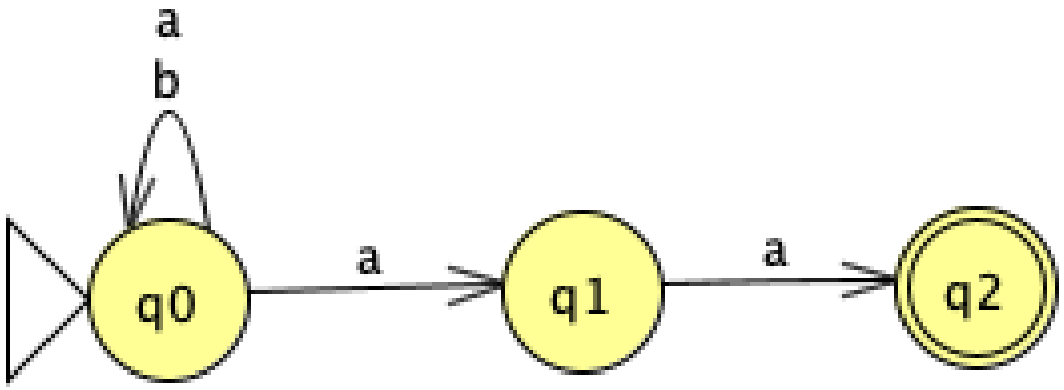
NFA Example 7

- Consider the following NFA.
- Is aaaaa in the language?
- Yes, we can find at least one way of ending to q2.



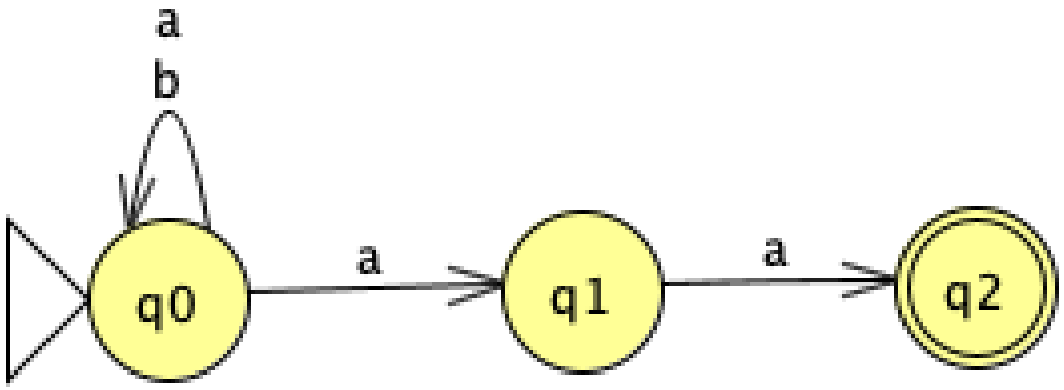
NFA Example 7

- Consider the following NFA.
- Is ababaa in the language?
- Yes, we can find at least one way of ending to q2.



NFA Example 7

- Consider the following NFA.
- Is ababa in the language?
- No way to get to a final state.



Every NFA can be converted to a DFA

- if we have an NFA M1, then we can construct a DFA M2 such that M1 and M2 accept the same set of words.
- While NFAs can have fewer states, the transitions are more complicated than DFAs.

Constructing a DFA from an NFA

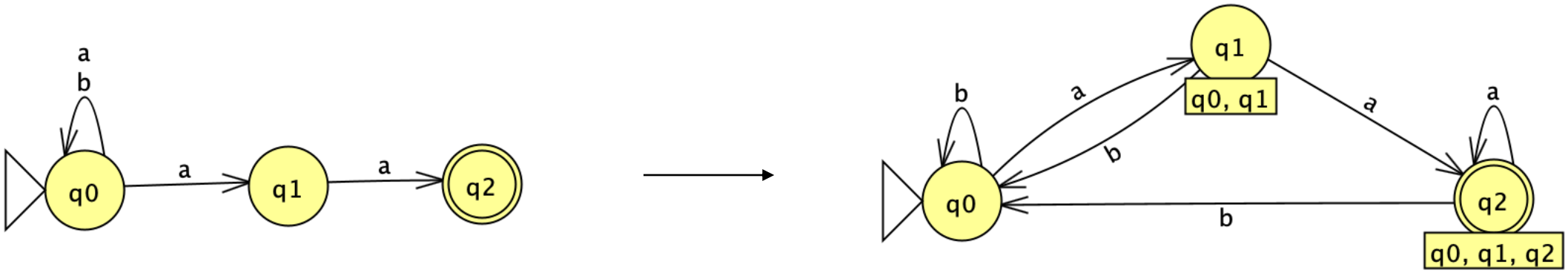
- Each state in the DFA is a subset of the states in the NFA.
- The start state of the DFA is the set containing only the start state of the NFA.
- The final state of the DFA is all of its states which have the final state of the NFA as elements.
- Transitions are more complicated as the output state of any DFA state for any given input is the *union* of output states of all of the NFA states of that DFA state.
- How many DFA states can we have at most?
  - $2^n - 1$ where n is the number of NFA states.
  - Think of each DFA state like a n bit number
    - 1 if it represents the original NFA state, 0 otherwise.
    - Can't have all zeros, so $2^n - 1$

Constructing a DFA from an NFA

- Create the start state (q_0) and add it to queue.
  - In a queue, the first in is the first out (FIFO, the opposite idea to LIFO in stacks).
- As long as queue isn't empty:
  - Remove front state s from queue
  - $s_{new} = []$
  - for each letter l in the alphabet
    - if any "old state", q_i, in s has a transition $q_i : l \rightarrow q_j$
      - add q_j to s_{new}
      - if s_{new} doesn't exist already, create state s_{new} and add $s_{new}$to queue q
  - if any states don't have transitions for all letters in the alphabet, create a "sink" state that transitions to itself on all letters and have states transition to here for any remaining alphabet letters

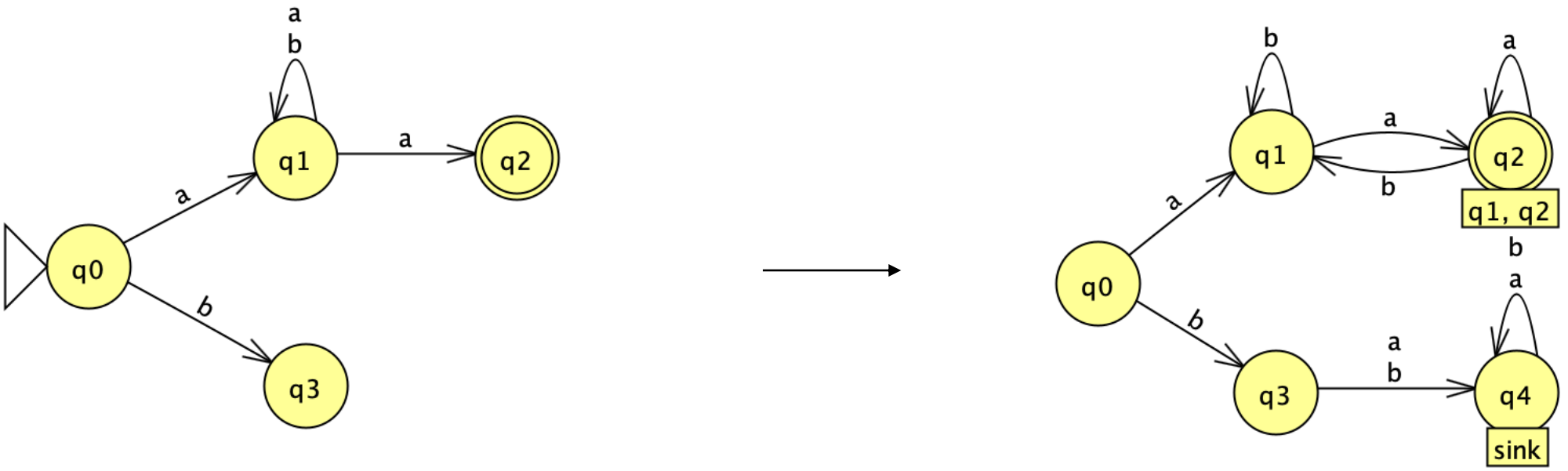
Convert example 4 from NFA to DFA

- Accepts strings of a's that end in 2 a's
- (a|b)*aa



Convert example 2 from NFA to DFA

- Accepts strings of a's that start and end with a
- a(a|b)*a



Regular languages

- A **regular language** is a language that can be described by a DFA (or an NFA, remember, they're equivalent).
- A regular language is also any language that can be described by a regular expression!

Regular languages

- Here is an example of a non-regular language
- 0^n 1^n for any n, that is the language of some number of zeros followed by the *same* number of 1s.
- Can you come up with a regular expression for this language?
  - seems hard, since there's no tool for us to count
- Can you come up with a DFA or NFA for this language?
  - would have to have 2^(n+1) states and states are the only way we can count.
- only problem is that n isn't finite!
  - Consider any DFA that recognizes strings of 0^n 1^n for some fixed n.
  - It won't recognize string 0^(n+1) 1^(n+1)
- Basic idea behind pumping lemma (more in CS101)

Practice Time

- Which of these languages are regular?
  - Language consisting of repetitions of either 01 or 10
  - Language of strings with an equal number of 0s and 1s
  - Language over a's and b's of alternating a's starting with an a, i.e. every other character in the string is an a
  - Language of all words that are palindromes, i.e. read the same forward and reversed

Answer

- Which of these languages are regular?
  - Language consisting of repetitions of either 01 or 10
    - Regular: (01|10)*
  - Language of strings with an equal number of 0s and 1s
    - Not regular
  - Language over a's and b's of alternating a's starting with an a, i.e. every other character in the string is an a
    - Regular: a((a|b)a)*|a((a|b)a)*(a|b)
  - Language of all words that are palindromes, i.e. read the same forward and reversed
    - Not regular

JFLAP examples:

- NFA examples